Ronald W. Sudol III

SNHU CS320

Project 2

06/18/2023

### **Summary and Reflections Report**

1. **Summary**
   1. Describe your unit testing approach for each of the three features.
      1. To what extent was your approach **aligned to the software requirements**? Support your claims with specific evidence.

**<** My approach to unit testing for each of the three features was aligned to the software requirements because I developed the unit tests incrementally after each developing each functionality. For instance, in the contact class I first developed the code to create a contact object and then developed code to test that the object was successfully created. In the task service class, I first developed code to add a task to a list and then developed tests to ensure that the task was successfully added to the list. In the appointment service class, I developed code to delete and appointment and then developed code to ensure that an appointment was successfully deleted from a list. **>**

* + 1. Defend the overall quality of your JUnit tests. In other words, how do you know your JUnit tests were **effective** based on the coverage percentage?

**<** I know that my unit tests were effective based on coverage percentage because of the incremental approach that I took in developing the test with the code. This ensured that I missed as few aspects of the codes testability as possible. Further, running the code coverage in Eclipse showed greater than 80% coverage on all .java files. **>**

* 1. Describe your experience writing the JUnit tests.
     1. How did you ensure that your code was **technically sound**? Cite specific lines of code from your tests to illustrate.

**<** Developing code incrementally also helped to ensure that my code was technically sound. For instance, I would first write this portion of Task.java:

// if inputs validated, create Task object

this.taskId = id;

this.taskName = name;

this.taskDescription = description;

After checking the about code for syntax errors, I would make test cases in in the TaskTest.java file like this:

//test object creation

assertTrue(testTask.getId().equals("123"));

assertTrue(testTask.getName().equals("Homework"));

assertTrue(testTask.getDescription().equals("Complete Mile Stone Two!"));

By enuring that the code compiled free of syntax errors and that it tested properly, I could be fairly certain that it was techically cound before moving on the the next portion of the code. **>**

* + 1. How did you ensure that your code was **efficient**? Cite specific lines of code from your tests to illustrate.

**<** I made sure that my code was efficient by grouping similar functions. This allowed me to ensure that I met all required functionality and didn’t miss anything. For instance all the tests for null input when creating a task object are grouped as follows:

// null tests

@Test

void testTaskIdNull() {

//test for null id

Assertions.assertThrows(IllegalArgumentException.class, () -> {

new Task(null,"Homework","Complete Mile Stone Two!");

});

}

@Test

void testTaskNameNull() {

//test for null name

Assertions.assertThrows(IllegalArgumentException.class, () -> {

new Task("123",null,"Complete Mile Stone Two!");

});

}

@Test

void testTaskDescriptionNull() {

//test for null description

Assertions.assertThrows(IllegalArgumentException.class, () -> {

new Task("123","Homework",null);

});

}

**>**

1. **Reflection**
   1. Testing Techniques
      1. What were the **software testing techniques** that you employed in this project? Describe their characteristics using specific details.

**<**  Unit tests are a software testing technique where individual components of a larger working program are designed and tested as smaller individual pieces known as units. I employed the unit tests reiteratively as I developed the source code. This ensured that each section of code was developed in a manner that is technically sound and ensured that no aspects of the source codes functionality were missed by the unit tests. For instance, for the appointment features, I developed the code that creates an appointment object. I then developed the code to test that the object was properly initialized. I unit tested the code in this way before moving on which helped me be sure that my code was sounds and that the functionality was accounted for in the test case file. **>**

* + 1. What are the **other software testing techniques** that you did not use for this project? Describe their characteristics using specific details.

**<**  Other software techniques that we had not used as much would include integration testing. Integration testing would test the interaction between individual components. For instance, if a displayable calendar were to be added to the project, the calendar itself would be developed and tested as a unit in a calendar class. Later integrations tests would be run to make sure that the appointment class was creating and correctly passing appointment objects that could be used and displayed by the calendar class. While, individually, the units may be testably sound, integration testing would ensure that the data flow and interaction between those two units is also testably sound. **>**

* + 1. For each of the techniques you discussed, explain the **practical uses and implications** for different software development projects and situations.

**<**  Unit tests are more practically employed early in the development phase. This ensures that the product is being built using secure and sound code. Integration testing is more practically employed a bit later in the development lifecycle, after units have already been developed. The implication here is that while these different techniques may be employed at different points in the development lifecycle, they actually work together to ensure a sound product. This is because the unit tests first ensure that individual pieces of a program are sounds and then integration tests ensure the individual pieces fit together in a way that is sound. **>**

* 1. Mindset
     1. Assess the mindset that you adopted working on this project. In acting as a software tester, to what extent did you employ **caution**? Why was it important to appreciate the complexity and interrelationships of the code you were testing? Provide specific examples to illustrate your claims.

**<** In switching to the mindset of a software tester, I employed caution in considering the various features of each class, the functional purpose of each feature, the kinds of data and variables handled by each feature, and the way in which each object would interact with its respective service feature. It was important to consider the complexity and interrelationship of these things because it effected test coverage. For instance, the getters/setters for an object class were declared and defined in the object class but they were implemented in the service class. Because of this, if one ran a coverage report on just the object class file, the report would show a lower coverage percentage than if the coverage report was run on the entire project. This is because my tests for the setters were defined in the service class's JUnit Test case file. **>**

* + 1. Assess the ways you tried to limit **bias** in your review of the code. On the software developer side, can you imagine that bias would be a concern if you were responsible for testing your own code? Provide specific examples to illustrate your claims.

**<** One way I tried to eliminate bias was by taking breaks and walking away from the code for a bit after developing part of a feature in hopes of having “fresh eyes” when I returned. One can easily get tunnel vision and start to assume that just because the code compiles that it must automatically be logically and technically sound. An example is that that I initially wrote code that simply test the phone number string in the Contact class was ten characters in length but failed to check that the characters were digits. I assumed because the code compiled so it must be sound but I initially failed to consider that ten letters would pass the tests and but would not meet the required functionality. **>**

* + 1. Finally, evaluate the importance of being **disciplined** in your commitment to quality as a software engineering professional. Why is it important not to cut corners when it comes to writing or testing code? How do you plan to avoid technical debt as a practitioner in the field? Provide specific examples to illustrate your claims.

**<** Discipline is crucial in the commitment to quality. Cutting corners can lead to released defects and escapements that can cost immense amounts of time and money or even human life. Testing every possible outcome of a program may create too much technical debt to be possible or practical. The answer to this is a systematic approach that focuses on first performing testing on the most crucial or functionally important features of a project. Then sequentially perform testing on remaining features in a pre-defined order of tiers based on descending importance and priority. This increases the overall coverage while ensuring the most important aspects are functional before creating unmanageable technical debt. **>**

**CITATIONS:**

Morgan, P., & Thompson, G. (2019b). *Software Testing: An ISTQB-BCS Certified Tester Foundation Guide*. BCS, The Chartered Institute for IT.

Garcia, B. (2017). *Mastering Software Testing with JUnit 5*.